Applied Soft Computing 47 (2016) 251–261

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Parametric system identification using neural networks

Tarek A. Tutunji

Mechatronics Engineering Department, Philadelphia University, Jordan

ARTICLE INFO

Article history:
Received 25 November 2013
Received in revised form 4 March 2016
Accepted 8 May 2016
Available online 12 May 2016

Keywords:
Neural networks
Transfer functions
System identification
System response
ARMA models

ABSTRACT

Neural networks are used in many applications such as image recognition, classification, control and system identification. However, the parameters of the identified system are embedded within the neural network architecture and are not identified explicitly. In this paper, a mathematical relationship between the network weights and the transfer function parameters is derived. Furthermore, an easy-to-follow algorithm that can estimate the transfer function models for multi-layer feedforward neural networks is proposed. These estimated models provide an insight into the system dynamics, where information such as time response, frequency response, and pole/zero locations can be calculated and analyzed. In order to validate the suitability and accuracy of the proposed algorithm, four different simulation examples are provided and analyzed for three-layer neural network models.

1. Introduction

Artificial Neural Networks (ANN) are mathematical models that are used to imitate the biological neurons in the brain. They are used as black box models to identify unknown functions by mapping input–output data. Many books have been written on the use of ANN in identification and control applications. Haykin [1], collected and established solid foundations in the theory of ANN. Liu [2] presented various ANN structures and discussed their applications in nonlinear identification and control systems (adaptive and predictive). Norgaard et al. [3] described different approaches to ANN-based identification and control of dynamic systems. Zilouchin with Jamshidi [4] collected and edited several articles that were concerned with the theory and applications of intelligent controllers. Furthermore, Demuth et al. [5] described different ANN control architectures (model predictive control, NARMA-L2 control, and model reference control) that were used in the ANN Toolbox guide for MATLAB.

Isermann and Munchhof [6] published a well-structured and comprehensive book entitled *Identification of Dynamic systems* where they described and compared many system identification methods. In their description of neural networks, they made the following statement *"Their main disadvantage is the fact that for most neural networks, the net parameters can hardly be interpreted in a physical sense, making it difficult to understand the results of the modeling process (Page 19)"*. They reiterated the statement again *"The main disadvantage is the fact that the resulting models cannot be interpreted well as the structure of the neural nets in general does not allow a physical interpretation (Page 534)"*. This paper is concerned with transforming the ANN model into transfer function model and therefore providing an insight into the physical system behavior.

Neural networks have been used extensively in identifying dynamic systems in different publications. Efe and Kaynak [7] studied and compared different ANN structures used in the identification of nonlinear systems. Liu et al. [8] used Volterra polynomial basis function neural networks for on-line identification of nonlinear systems. Gabrijel and Dobnikar [9] used recurrent neural networks for on-line identification and reconstruction. These researchers showed the capability of neural networks to identify systems.

Sahoo et al. [10] used different neural network model structures (polynomial and trigonometric expansions) to identify nonlinear autoregressive models. The functional expansions were used to capture the delayed input–output data which were then multiplied by the network weights and used as inputs to a hyper-tangent activation function. They proposed a robust $H\infty$ filter learning algorithm to update the network weights. They used simulated nonlinear time-varying plants to show that their proposed algorithm provides lower mean-square-error than forgetting factor recursive least squares algorithm, especially when noise is added. However, the converged ANN weights were not compared to the simulated plants' parameters.

Coban [11] proposed a recurrent neural network with added context layer for dynamic system identification. The proposed network architecture was constructed using a general feedforward network, but with added 'special' hidden layer that interacts exclu-

E-mail addresses: ttutunji@yahoo.com, ttutunji@philadelphia.edu.jo

http://dx.doi.org/10.1016/j.asoc.2016.05.012
1568-4946/© 2016 Elsevier B.V. All rights reserved.

sively with the 'original' hidden layer. Dynamic backpropagation was used to train the network and update its weights. The work was validated using linear and nonlinear simulated plants and experimental DC motor and showed that the results of the proposed network provided better performance than the Elman recurrent network. However, the converged network weights were not analyzed nor were they compared to the original plants' parameters.

Deng [12] proposed a series-parallel hybrid structure with two neural networks where one network was used to generate the desired plants' outputs that were used for training the second network. Such hybrid structure can improve the mapping capability of the networks and tested the proposed model on experimental 3D crane system to demonstrate the validity of his work. However, the relationship between the network weights and the crane's transfer function was not studied.

Darus and Al-Khafaji [13] used neural networks for non-parametric identification of flexible plates. They conducted laboratory experiments and validated their work by comparing the model response with the measured data. They also performed correlation tests with the multi-layer perceptron neural, adaptive Elman networks, and adaptive neural fuzzy networks. However, their interest was concerned with nonparametric identification only.

Han et al. [14] investigated an automatic self-organizing neural network that adapted its architecture (number of neurons and topology) during the training process in order to improve the network performance. They provided the pseudo code of the *adaptive connecting and pruning algorithm* and *feedforward computation* used. They performed simulations of nonlinear models to compare their algorithm results with other adaptive networks and showed that the proposed algorithm provided better performance in CPU time, mean-square-error, and average-percentage error. However, they did not elaborate on the relationship between the network weights and the simulated plant's parameters.

Xie et al. [15] developed an identification method using ANN based on Bouc–Wen differential model to identify memory-type nonlinear hysteretic systems. They conducted laboratory experiments to identify the restoring force of wire cable vibration isolation system. They were able to identify the parameters of the Bouc–Wen model, but they did not relate these parameters to the plant's transfer function.

All the previously described publications succeeded in developing ANN architectures, learning algorithms, and mathematical models for system identification applications. However, none of these researchers provided a clear mathematical relationship between the network weights and the identified systems in parametric format.

Another closely-related application that researchers worked on was time-series forecasting. Khashei and Bijari [16] used neural network models for time-series forecasting while Zhang [17] used a hybrid Auto-Regressive-Moving-Average and neural network model for time-series forecasting. Again, these researchers did not provide the mathematical relationship between the network weights and the estimated functions.

From the vast amount of research published in the area of neural network identification, only a few investigated the mathematical relationship between the network weights and the parameters of the identified systems. Fung et al. [18] derived equations for the frequency response and general transfer functions of multi-layer networks in terms of the network weights. They used series expansions and Volterra kernel within the network models to establish their equations. However, their work was general, very mathematically involved, and did not provide a clear path to follow. Chon and Cohen [19] did impressive work in estimating the parameters for linear and nonlinear Auto-Regressive Moving-Average (ARMA) models using neural network weights. They provided simulation results for several systems and compared the results between the network identification and least square ARMA identification. However, their work was restricted to polynomial activation functions and did not consider the frequency responses of their models. Lopez and Caicedo [20] used multilayer perceptron for parametric identification. They showed explicit equations for the linear activation cases, but they did not provide those equations for the nonlinear activation functions. Instead, they re-structured the error criteria and used Bayesian training to deal with the nonlinearities. In all of the described work, none provided a clear and easy-to-follow algorithm that shows how to relate the network weights to system's parameters.

Chen and Chen [21] discussed a neural-network-based system identification technique to determine the z-transfer function of a building envelope from experimental data. Neural networks were used to determine the Markov parameters of the process and Eigen-system realization algorithm was used to identify a minimal order state space presentation. However, the neurons were assumed to operate in the linear range only. Also, the work studied only the hyperbolic activation function and the simulation results were applied to the specific case of heat conduction through a wall.

Fei et al. [22] proposed a linear recurrent neural network and identified transfer function matrix models for multi-variable systems. Simulation results were provided to show that the proposed method can deliver the transfer function parameters from the neural network weights. However, the active functions of the hidden and output layers were linear and therefore the proposed method can only be applied in the identification of linear systems. They concluded that investigation is required to establish whether similar results can be found when nonlinear activation functions are used.

In previous work, Tutunji [23] presented a method to identify transfer functions for linear models using neural network weights with single layer only. This paper builds on those results and expands the work to include multi-layer neural network and nonlinear models.

The main contribution of the paper is the establishment of a clear relationship between the ANN weights and the transfer function parameters. Therefore, providing better interpretation (in a physical sense) where important information, such as frequency response and pole locations, can be explored. More importantly, a clear and easy-to-follow algorithm is provided that can transform the network results into ARMA models and therefore identify the system's transfer function.

In Section 2, theoretical background for system identification and neural network architecture is provided. Section 3 provides the mathematical derivations for the proposed method and describes the algorithm used. The simulation results are given in Section 4 and the conclusion is provided in Section 5.

2. Theoretical background

This section provides the background theory that is essential to the proposed algorithm and is divided into two parts: system identification and neural network architecture.

2.1. System identification

System identification is the process of using appropriate mathematical models and learning algorithms in order to map experimental data by minimizing an error criterion between the system's desired output and the model output.

Auto-Regressive Moving-Average (ARMA) models are linear regression models that use difference (or differential) equations to relate the model output to present inputs, past inputs and past

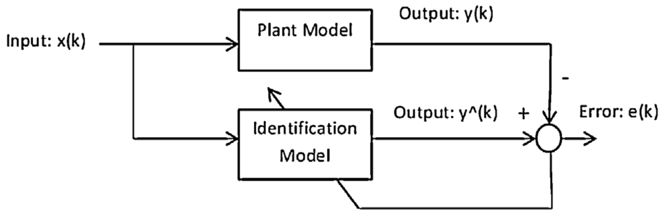

Fig. 1. General system identification block diagram.

outputs. A general discrete-time ARMA model is presented as:

$$y(k) = a_1 y(k-1) + \ldots + a_n y(k-n) + b_0 x(k) + \ldots + b_m x(k-m) \tag{1}$$

Which can also be presented as:

$$y(k) = \sum_{i=0}^{m} b_i x(k-i) + \sum_{j=1}^{n} a_j y(k-j) \tag{2}$$

where $x(k)$ and $y(k)$ are the model's input and output at sample k while b_i and a_j are the model's parameters. The transfer function can be found by using the z-transform as follows:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + \ldots + b_m z^{-m}}{1 - a_1 z^{-1} - \ldots - a_n z^{-n}} \tag{3}$$

Eq. (2) can be put in vector format as $y(k) = \theta^T \emptyset(k)$ where $\theta^T = [a_1, \ldots a_n, b_0, \ldots, b_m]$ is the parameter vector and $\emptyset(k) = \left[y(k-1), \ldots y(k-n), x(k), \ldots, x(k-m)^T\right]$ is the measurement vector [6,24]. Parametric system identification approximates the parameters, a_j and b_i, to find an estimated parameter vector $\hat{\theta}^T = \left[\hat{a}_1, \ldots \hat{a}_n, \hat{b}_0, \ldots, \hat{b}_m\right]$ which can be used to calculate an estimated output $\hat{y}(k) = \theta^T \emptyset(k)$. The goal is to find the appropriate estimated parameters to minimize the error between the desired output $y(k)$ and estimated output $\hat{y}(k)$. Fig. 1 shows the general block diagram for system identification.

Algorithms, such as Least Square (LS), Recursive Least Squares (RLS), Recursive Prediction Error Method (RPEM) are well-established in literature [6,24] and can be used for such purposes. In previous work, impulse response data within ARMA models were used to identify mechatronic systems [25] with good success.

Many physical systems are nonlinear in nature and therefore require nonlinear modeling. However, the many structural possibilities of non-linear relations between the input and output of dynamic systems makes it difficult to identify many types of these systems with accurate parametric models. Identification of such models can involve advanced concepts such as Volterra series and Hammerstein models [26].

Another option is to modify the ARMA model to include non-linearity terms and create a Nonlinear ARMA model (NARMA) as:

$$y(k) = \sum_{i=0}^{m} b_i x(k-i) + \sum_{j=1}^{n} a_j y(k-j) + \sum_{i=0}^{m} \sum_{j=0}^{n} b_{ij} x(k-i) x(k-j) + \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} y(k-i) y(k-j) + \sum_{i=0}^{m} \sum_{j=1}^{n} c_{ij} x(k-i) y(k-j) \tag{4}$$

These models add new parameter (a_{ij}, b_{ij}, and c_{ij}) and introduce difficulties in predicting the appropriate order for each summation term. Moreover, these models do not follow the traditional transfer function provided in Eq. (3) and therefore are more difficult to deal with. These obstacles make neural networks an appropriate choice for modeling nonlinear systems.

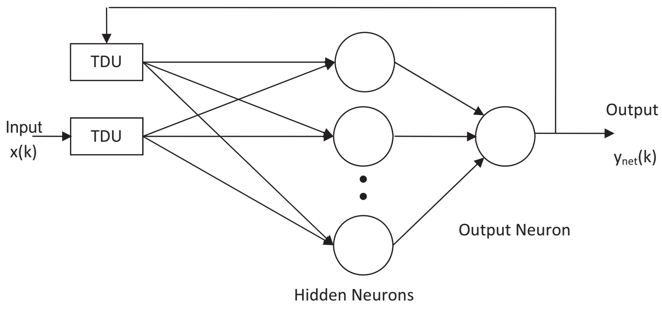

Fig. 2. General architecture for a dynamic network.

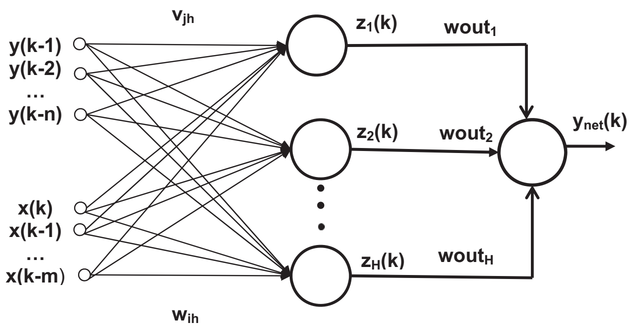

Fig. 3. Neural network architecture used.

2.2. Neural network architecture

Neurons are information-processing units and are the basic elements in neural networks. ANN can be composed of multiple layers where each layer can be composed of several neurons. For feedforward networks, the information flows in one direction (i.e. input to output) while recurrent networks allows for feedback signals. For the special case where all the neural network's inputs are composed of delayed inputs and delayed network's output, the network can be referred to as dynamic network.

Fig. 2 shows a dynamic network with one 'main' input and one output, $x(k)$ and $y_{\text{net}}(k)$ respectively, where k is the sample number. The TDU is a Time Delay Unit, equivalent of Z^{-n}, which results in delayed samples. Therefore, the network can be used as multi-inputs where delayed I/O samples are fed to the network as inputs. The delay will depend on the model order.

Another option is to use the delayed plant's output (instead of the delayed network's output) as the feedback to the network. Therefore, the network's output becomes a function of past inputs and outputs as given in the following equation:

$$y_{\text{net}}(k) = f(x(k), x(k-1), \ldots, y(k-1), y(k-2), \ldots) \tag{5}$$

This is considered a feedforward network with multi-inputs $y(k-1) \ldots y(k-n)$, $x(k), \ldots, x(k-m)$, as shown in Fig. 3. These input samples are the result of the TDU where each input is multiplied by its corresponding weight, such that v_{jh} connects input $y(k-j)$ with hidden neuron h and w_{ih} connects input $x(k-i)$ with hidden neuron h. The hidden nodes have outputs $z_1(k) \ldots z_h(k)$ which are in-turn multiplied by output weights $wout_1 \ldots wout_H$ to provide the network's output $y_{\text{net}}(k)$. The network's output is written as $y_{\text{net}}(k)$ to differentiate it from the plant's output $y(k)$ as described in the previous section.

Bias weights were not used in this network in order to avoid introducing new inputs (i.e. the objective was to limit the network's inputs to $x(k)$ and delayed samples of $x(k)$ and $y(k)$ so as to mimic the I/O in a general discrete-time systems).

Table 1
Neuron activation functions with Taylor expansion result.

Function name	Math function $f(x)$	Function derivative $f'(x)$	1st two terms of the Taylor expansion at 0	Approximation result
Hyperbolic tangent with $\alpha = 1$	$\frac{1-e^{-x}}{1+e^{-x}}$	$0.5\left(1-f^2(x)\right)$	$f(0)+0.5\left(1-f^2(0)\right)x$	$0.5x$
Hyperbolic tangent with $\alpha = 2$	$\frac{1-e^{-2x}}{1+e^{-2x}}$	$1-f^2(x)$	$f(0)+\left(1-f^2(0)\right)x$	x
Sigmoid	$\frac{1}{1+e^{-x}}$	$f(x)(1-f(x))$	$f(0)+f(0)(1-f(0))x$	$0.5+0.25x$
Gaussian	$e^{-(x-c)^2}$	$-2(x-c)e^{-(x-c)^2}$	$e^{-c^2}+2ce^{-c^2}x$	c_1+c_2x

The network's mathematical operations are:

$$y_{\text{net}}(k) = g\left(\sum_{h=1}^{H} z_h(k)\text{wout}_h\right) \tag{6}$$

$$z_h(k) = f\left(\sum_{i=0}^{m} w_{ih}x(k-i) + \sum_{j=1}^{n} v_{jh}y(k-j)\right) \tag{7}$$

Here, f and g are the activation functions for the hidden and output nodes respectively. There are several activation functions that can be used such as the sigmoid and the hyperbolic tangent [1].

The goal is to find the optimal network weights (i.e. v_{jh}, w_{jh} and $wout_h$) in order to minimize an error criterion (between the network's output, $y_{\text{net}}(k)$, and the desired plant output $y(k)$). One commonly used error is the Sum Square Error (SSE),

$$\text{SSE} = \sum\nolimits_{k=1}^{K}(e(k))^2 = \sum\nolimits_{k=1}^{K}(y_{\text{net}}(k) - y(k))^2 \tag{8}$$

Several algorithms, such as the backpropagation with Levenberg–Marquardt training, can be used to train the network (i.e. find the optimal weights) [6]. The Levenberg-Marquardt approximates 2nd derivative (i.e. the Hessian) using 1st order derivatives such as:

$$\text{NewWeights} = \text{OldWeights} - \left[J^TJ + \mu I\right]^{-1} J^T e \tag{9}$$

Here, e is a vector of network errors (difference between network and desired outputs with elements $e(k)$ as described in Eq. (8)), I is the identity matrix, and J is the Jacobian matrix (first derivatives of the network errors with respect to the weights). Details of the Jacobian matrix and error vector can be found in [27]. For the output layer, the Jacobian elements are calculated from:

$$\frac{\partial e}{\partial \text{wout}_h} = \frac{\partial e}{\partial y_{\text{net}}}\frac{\partial y_{\text{net}}}{\partial \text{wout}_h} \tag{10}$$

As for first layer weights (i.e. v_{jh} and w_{ih}), the Jacobian matrix can be computed using standard backpropagation technique:

$$\frac{\partial e}{\partial w_{ih}} = \frac{\partial e}{\partial y_{\text{net}}}\frac{\partial y_{\text{net}}}{\partial z_h}\frac{\partial z_h}{\partial w_{ih}} \tag{11}$$

$$\frac{\partial e}{\partial v_{jh}} = \frac{\partial e}{\partial y_{\text{net}}}\frac{\partial y_{\text{net}}}{\partial z_h}\frac{\partial z_h}{\partial v_{jh}} \tag{12}$$

This paper is not concerned with the weight convergence, but rather with using the converged network weights to approximate the original system's transfer function. Therefore, no detailed discussion will be provided for training the network weights. This is well-established in the literature [1].

Neural networks have been used successfully in identifying linear and nonlinear systems. This is due to the fact that neural networks have good approximation capabilities and adaptive features. However, the final neural network model is rarely related to the transfer function parameters (i.e. poles and zeros).

3. Proposed NN2TF algorithm

In the previous sections, mathematical presentations for ARMA models and neural network were provided. In this section, a mathematical relationship between the ARMA and ANN models is derived and a Neural Network to Transfer Function (NN2TF) algorithm is provided.

A common activation function used in neural networks is the hyperbolic tangent that provides smooth output between −1 and +1 and is easy to differentiate. Therefore, the output of the hidden node, from Eq. (7), becomes:

$$z_h(k) = \frac{1 - e^{-\alpha \text{net}_h(k)}}{1 + e^{-\alpha \text{net}_h(k)}} \tag{13}$$

$$\text{net}_h(k) = \sum\nolimits_{i=0}^{m} w_{ih}x(k-i) + \sum\nolimits_{j=1}^{n} v_{jh}y(k-j) \tag{14}$$

The following results are derived for the special case where the activation function at the hidden nodes, f in Eq. (7), is the hyperbolic tangent (with $\alpha = 1$) and the activation function at the output node, g in Eq. (6), is the linear summation.

Taylor expansion can be used to approximate Eq. (13) about point '0' (i.e. $\text{net}_h(k) = 0$). Note that the hyperbolic tangent range is between −1 and 1 and therefore approximation around the mid-point (i.e. zero) is reasonable.

$$z_h(k) = \frac{1-e^0}{1+e^0} + \frac{2e^0}{\left(1+e^0\right)^2}(\text{net}_h(k) - 0) + \frac{-2e^0\left(1-e^0\right)}{\left(1+e^0\right)^3}(\text{net}_h(k) - 0)^2 + \ldots \tag{15}$$

To avoid the nonlinearities, only the first two terms are used and Eq. (15) is simplified to:

$$z_h(k) = 0.5\sum\nolimits_{i=0}^{m} w_{ih}x(k-i) + 0.5\sum\nolimits_{j=1}^{n} v_{jh}y(k-j) \tag{16}$$

The output neuron gives:

$$y_{\text{net}}(k) = \sum\nolimits_{h=1}^{H}(\text{wout}_h z_h(k)) \tag{17}$$

Substituting Eq. (16) in Eq. (17) results in:

$$y_{\text{net}}(k) = 0.5\sum_{h=1}^{H}\left(\text{wout}_h\left(\sum_{i=0}^{m} w_{ih}x(k-i) + \sum_{j=1}^{n} v_{jh}y(k-j)\right)\right) \tag{18}$$

By comparing Eq. (18) with Eq. (2), the ARMA parameters can be estimated as:

$$\hat{a}_j = 0.5\left(\sum_{h=1}^{H}\left(\text{wout}_h v_{jh}\right)\right) \tag{19}$$

$$\hat{b}_i = 0.5\left(\sum_{h=1}^{H}(\text{wout}_h w_{ih})\right) \tag{20}$$

Table 2
ARMA parameter estimators for different activation functions.

Function	$\hat{a}_j$	$\hat{b}_i$	Offset
Hyperbolic tangent, $\alpha = 1$	$0.5\sum_{h=1}^{H}\left(\text{wout}_h v_{jh}\right)$	$0.5\sum_{h=1}^{H}\left(\text{wout}_h w_{ih}\right)$	–
Hyperbolic tangent, $\alpha = 2$	$\sum_{h=1}^{H}\left(\text{wout}_h v_{jh}\right)$	$\sum_{h=1}^{H}\left(\text{wout}_h w_{ih}\right)$	–
Sigmoid	$0.25\sum_{h=1}^{H}\left(\text{wout}_h v_{jh}\right)$	$0.25\sum_{h=1}^{H}\left(\text{wout}_h w_{ih}\right)$	$0.5\sum_{h=1}^{H}\left(\text{wout}_h\right)$
Guassian	$c_2\sum_{h=1}^{H}\left(\text{wout}_h v_{jh}\right)$	$c_2\sum_{h=1}^{H}\left(\text{wout}_h w_{ih}\right)$	$c_1\sum_{h=1}^{H}\left(\text{wout}_h\right)$

These equations establish a mathematical relationship between the network weights and the ARMA parameters. Therefore, the network information can be used for parametric identification (i.e. approximate transfer function).

Similar derivations can be carried out when different activation functions are used for the hidden layer. Table 1 shows different activation functions with their derivatives, Taylor expansions, and approximation results.

Substituting the approximated results displayed in Table 1 results in Eq. (17) produces the following equations for the hyperbolic tangent (with $\alpha = 2$), sigmoid, and Gaussian:

$$y_{\text{net}}(k) = \sum_{h=1}^{H}\left(\text{wout}_h\left(\sum_{i=0}^{m} w_{ih}x(k-i) + \sum_{j=1}^{n} v_{jh}y(k-j)\right)\right) \tag{21}$$

$$y_{\text{net}}(k) = \sum_{h=1}^{H}\left(\text{wout}_h\left(0.5 + 0.25\sum_{i=0}^{m} w_{ih}x(k-i) + 0.25\sum_{j=1}^{n} v_{jh}y(k-j)\right)\right) \tag{22}$$

$$y_{\text{net}}(k) = \sum_{h=1}^{H}\left(\text{wout}_h\left(c_1 + c_2\sum_{i=0}^{m} w_{ih}x(k-i) + c_2\sum_{j=1}^{n} v_{jh}y(k-j)\right)\right) \tag{23}$$

Note that the sigmoid and Gaussian functions have an added term (first term in Eqs. (22) and (23)) that is not multiplied by either the inputs or outputs of the system. These terms will not be part of the transfer function parameters, but rather a DC offset (as will be demonstrated in Example 2 in the simulation section). Using these equations, the estimated ARMA results were derived and displayed in Table 2.

Stability of ANN have been discussed in many publications. In recent work, Fu et al. [28] presented a robust ANN identifier that used Lyapunov function and singularity perturbed methods to guarantee bounded errors and weights. Gil et al. [29] presented an affine three-layer state-space ANN with global stability conditions based on Lyapunov stability theory. In general, learning algorithms will provide stable ANN models as long as they keep the network weights bounded within acceptable limits. The discussion of ANN stability is not within the scope of this work. However, the derived approximations in Table 2 do not guarantee stability (i.e. stable ANN model can results in unstable ARMA model) as explained next.

The stability of the discrete-time ARMA models, shown in Eq. (2), depend of the value of the a_j parameters. These a_j parameters determine the pole location of the transfer function shown in Eq. (3). The system would be unstable if the poles fall outside the unit circle. Therefore, it is possible to find an ANN model with bounded-weights that transforms into unbounded ARMA model. Such a model is provided in the simulation section for Example 4. A simple way to resolve this issue is to move the poles inside the unit circle by reducing the poles' magnitude without changing their angle as follows:

```
// Pseudo Code for stabilizing the approximated transfer function
poles = roots( [ 1 –$a_1$ –$a_2$ … -$a_n$])
for j= 1:n
        while ( abs(poles(j)) ) > 1.0
                pole(j)  = 0.9 * poles(j)
        end
end
// End of Code
```

Fig. 4 shows a complete step-by-step description of the propose algorithm. The algorithm can be divided into two stages: ANN training and transfer function transformation. The steps shown in the first stage are for general ANN training which can be implemented using Neural Network Toolbox codes (in MATLAB) or the researcher's developed program. The ANN structure should be composed of three layers (input, hidden, and output) with an activation function (chosen from Table 1) for the hidden layer and linear activation function for the output layer.

The second stage specify the proposed NN2TF transformation where Table 2 is used to transform the network weights to ARMA parameters. Then, the described pseudo code is used to adjust the model poles and the a_j parameters. Finally, the transfer function is created from the ARMA model (as shown in Eqs. (2) and (3)).

The proposed algorithm described in Fig. 4 shows how to transform an ANN model into an ARMA model. The advantage of this transformation is the ability to derive a transfer function from the network weights and therefore provide insight into the system under consideration.

All models contain errors and there is a tradeoff between model complexity and computation time. One way to test the models accuracy is through simulation runs by examining the resulting errors and overall system performance. Four simulation examples are provided in the next section in order to validate the proposed work.

4. Simulation results and discussion

The algorithm described in Section 3 was implemented using MATLAB. The simulation results are presented in this section and are divided into two parts: linear models and nonlinear models.

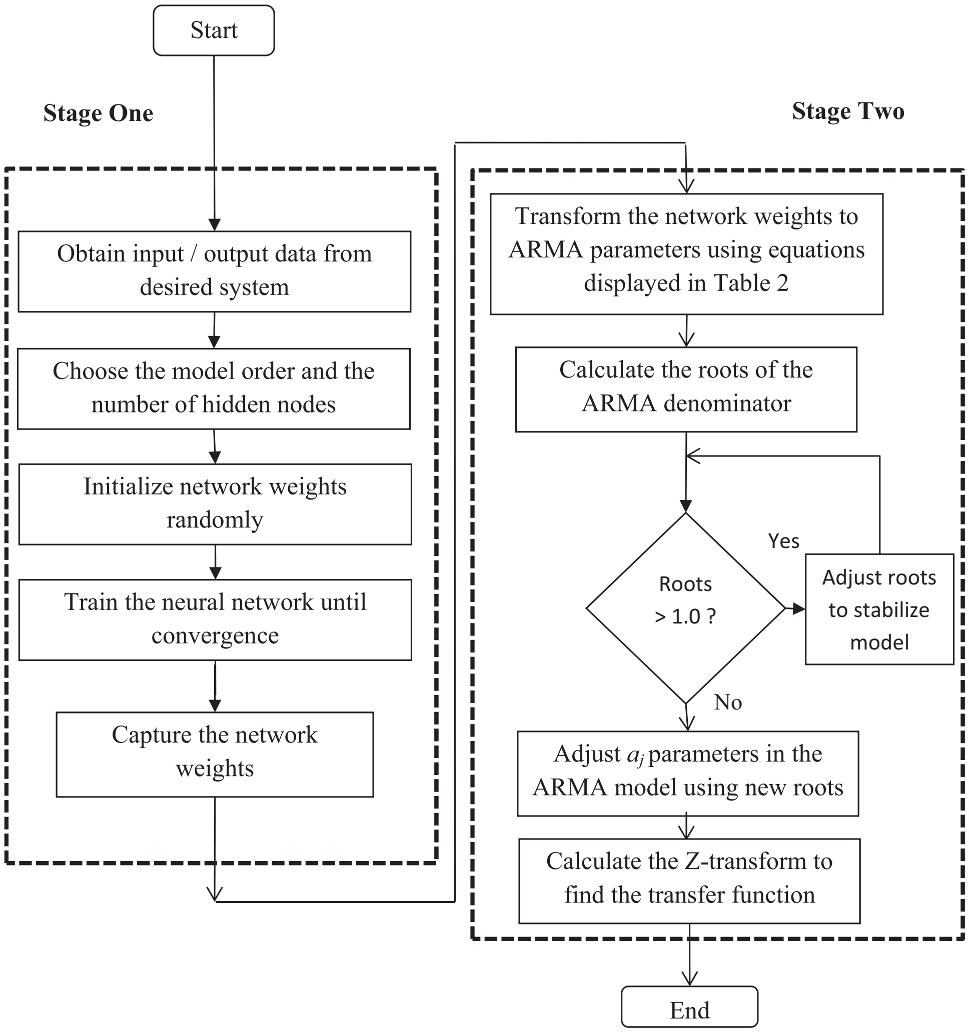
Start
Stage One
Obtain input / output data from desired system
Choose the model order and the number of hidden nodes
Initialize network weights randomly
Train the neural network until convergence
Capture the network weights
Stage Two
Transform the network weights to ARMA parameters using equations displayed in Table 2
Calculate the roots of the ARMA denominator
Roots > 1.0 ?
Yes
Adjust roots to stabilize model
No
Adjust a_j parameters in the ARMA model using new roots
Calculate the Z-transform to find the transfer function
End

Fig. 4. Flowchart for proposed NN2TF algorithm.

4.1. Linear models

Known ARMA models were used to test the proposed algorithm to validate the accuracy of the estimated model and to compare the original and estimated transfer functions.

The backpropagation was used to train the network. Other learning algorithms can be used that might give more accurate results. However, the scope of this work is not concerned with the network training, but rather with the transformation from the network weights to the system parameters.

A program was used to create random ARMA models where the parameters were generated from a normal distribution with a zero mean and a standard deviation of one. The input data was also randomly distributed with zero mean and standard deviation of one. A total of 60 runs were recorded and the results are displayed in Tables 3 and 4.

Table 3 displays the statistical data for the neural network model results where the network architecture and ARMA model were varied. The data sample size was varied between 25 and 100, the number of hidden neurons was varied between 3 and 5, and the model order was varied between 3, 4, and 5. These variations were used in order to test the proposed algorithm under different architectures and to provide appropriate statistical results. In all cases, the hypertangent tangent activation function was used in the hidden layer and a linear activation function was used in the output layer, as shown in Fig. 3.

The error criteria displayed in Table 3, SSE as described in Eq. (8), confirmed that the ANN was able to estimate the generated ARMA models with good accuracy as the results show that the SSE is very small. These results were expected as the success of ANN in identification applications is well-established. However, Table 3 was used as benchmark to validate the proposed method result displayed in Table 4.

The statistical results for the estimated transfer function, generated by the proposed algorithm described in Fig. 4, are shown in Table 4. Those are the same runs used in Table 3 (i.e. for each run, the converged network weights were transformed to the estimated parameters of the transfer function). Comparisons between Tables 3 and 4 show that the SSE was increased by magnitude of 10. This increase was expected because the proposed algorithm generated an approximate transfer functions using the ANN weights. Still, the proposed algorithm was able to provide estimated transfer functions that were able to follow the original transfer functions with small error differences. These results were mainly used to validate the accuracy of the proposed algorithm.

Because the system parameters and input data were all generated randomly, there was no need to add noise (i.e. the noise was already embedded in the system). However, the issue of noise will be considered in Example 4.

Table 3
SSE simulation results for the neural network model.

No. of samples	No. of hidden nodes	Model order	Min	Max	Average	Variance
25 samples	$H = 3$	Order = 3	2.74E – 04	2.10E – 02	6.59E – 03	7.05E – 05
		Order = 4	7.17E – 04	9.36E – 02	4.15E – 02	2.08E – 03
		Order = 5	5.20E – 04	6.40E – 03	3.62E – 03	5.80E – 06
	$H = 5$	Order = 3	7.98E – 04	2.86E – 02	7.42E – 03	1.42E – 04
		Order = 4	2.51E – 04	4.00E – 03	2.16E – 03	3.09E – 06
		Order = 5	2.10E – 04	4.58E – 02	1.08E – 02	3.84E – 04
100 samples	$H = 3$	Order = 3	2.50E – 03	1.21E – 01	5.72E – 02	1.83E – 03
		Order = 4	1.30E – 02	1.64E – 01	8.14E – 02	4.62E – 03
		Order = 5	2.40E – 03	3.97E – 01	1.42E – 01	2.29E – 02
	$H = 5$	Order = 3	1.05E – 04	1.33E – 02	4.43E – 03	3.40E – 05
		Order = 4	2.17E – 02	5.12E – 01	2.02E – 01	3.80E – 02
		Order = 5	1.50E – 03	1.81E – 01	7.49E – 02	5.47E – 03
Average			3.66E – 03	1.32E – 01	5.29E – 02	6.29E – 03

Table 4
SSE simulation results for the estimated transfer functions.

No. of Samples	No. of hidden nodes	Model order	Min	Max	Average	Variance
25 samples	$H = 3$	Order = 3	0.0011	0.4478	0.1533	0.0394
		Order = 4	0.0391	1.5265	0.6263	0.3816
		Order = 5	0.0177	2.5634	0.7010	1.1271
	$H = 5$	Order = 3	0.0173	0.1501	0.0657	0.0025
		Order = 4	0.0033	0.5537	0.1915	0.0484
		Order = 5	0.0026	2.0595	0.5198	0.7539
100 samples	$H = 3$	Order = 3	0.0061	0.4437	0.1548	0.0300
		Order = 4	0.0395	0.6880	0.2786	0.0740
		Order = 5	0.0058	2.1865	0.7021	0.7515
	$H = 5$	Order = 3	0.0004	0.0373	0.0104	0.0002
		Order = 4	0.1232	3.9365	1.5028	2.7400
		Order = 5	0.0086	2.2279	0.6111	0.8611
Average			2.21E – 02	1.40E + 00	4.60E – 01	5.67E – 01

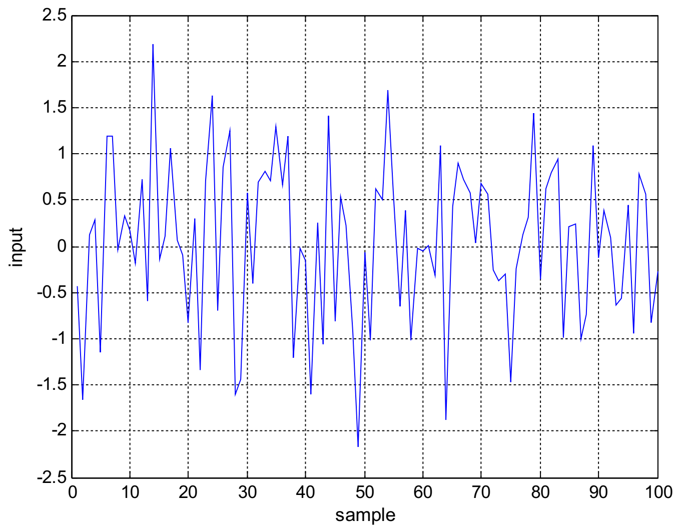

Fig. 5. Input samples used for 5th order system.

In order to have better insight into the proposed algorithm's results, one simulation run was explored in more details. The following example discusses the impulse, unit step, and bode responses of the estimated model. Other simulation runs displayed in Table 4 would provide similar accuracies between the original and identified models.

Example 1. The following 5th order transfer function model was generated to test the proposed algorithm.

Table 5
Parameter comparisons for 5th order models.

Denominator parameters	a_1	a_2	a_3	a_4	a_5
Original model	−0.2969	−0.5506	0.2466	−0.1297	0.0818
Estimated model	−0.3349	−0.5858	0.2540	−0.1308	0.0856
Numerator parameters	b_o	b_1	b_2	b_3	b_4
Original model	0.2341	0.0215	−1.0039	−0.9471	−0.3744
Estimated model	0.2444	0.0281	−1.0537	−1.0217	−0.4141

$$H(z) = \frac{0.2341z^5 + 0.0215z^4 - 1.0039z^3 - 0.94171z^2 - 0.3744z}{z^5 + 0.2969z^4 + 0.5506z^3 - 0.2446z^2 + 0.1297z - 0.0818} \tag{24}$$

A total of 100 input samples, as shown in Fig. 5, were generated and fed to the model and a neural network with three hidden nodes was used to identify the system. Once the ANN weights converged, the estimated ARMA parameters were calculated.

Table 5 displays the a_j and b_i parameters for the original system and the identified model while Table 6 shows the pole and zero locations for both models. It is clear from the tables that the algorithm was able to estimate the original system parameters with minimal error. This validates the proposed algorithm capability of using ANN weights to estimate the transfer function parameters and therefore confirms the algorithm's ability to perform parametric identification.

The input data, shown in Fig. 5, was used to excite the original and estimated models and the results are shown in Fig. 6. The original and estimated models provided similar outputs which further confirmed the accuracy of the estimated model.

Table 6
Pole and zero locations for 5th order system.

	Poles	Zeros	Gain
Original model	0.4361 $-0.4303 \pm 0.7306j$ $0.0637 \pm 0.5069j$	0 2.4467 −1.5877 $-0.4753 \pm 0.4311j$	0.2341
Estimated model	0.4369 $-0.4379 \pm 0.7499j$ $0.0520 \pm 0.5072j$	0 2.4526 −1.5931 $-0.4872 \pm 0.4430j$	0.2444

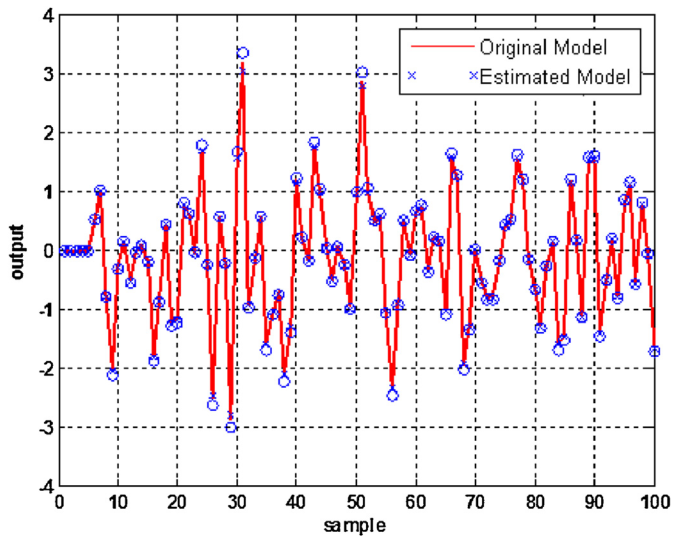

Fig. 6. Output samples for linear model example.

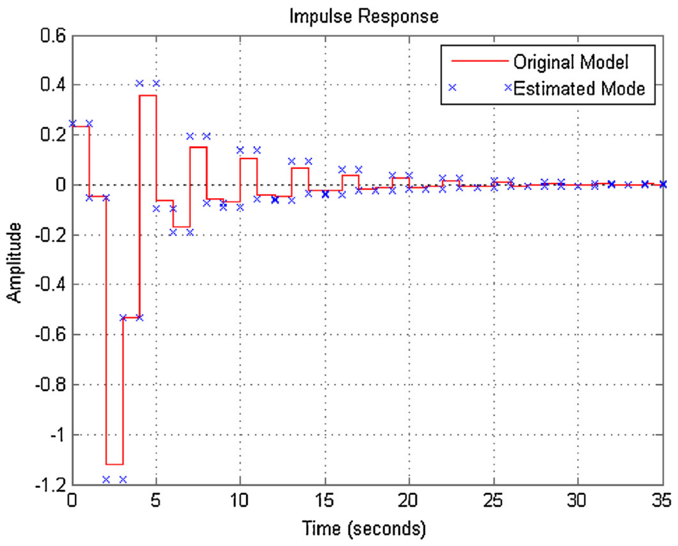

Fig. 7. Impulse response for linear model example.

Impulse and step responses are widely used in evaluating the plants behavior and are also used in designing appropriate controllers to meet the system's specifications. Therefore, an impulse and unit step signals were applied to the original and estimated models and their results are displayed in Figs. 7 and 8, respectively. In both figures, the estimated model's response was similar to the original model's response. This similarity in the behavior demonstrated the algorithm's effectiveness in duplicating the desired responses.

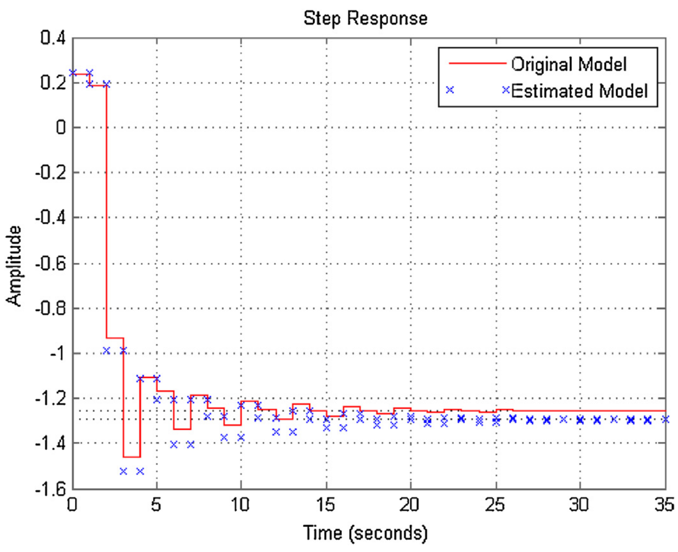

Fig. 8. Step response for linear model example.

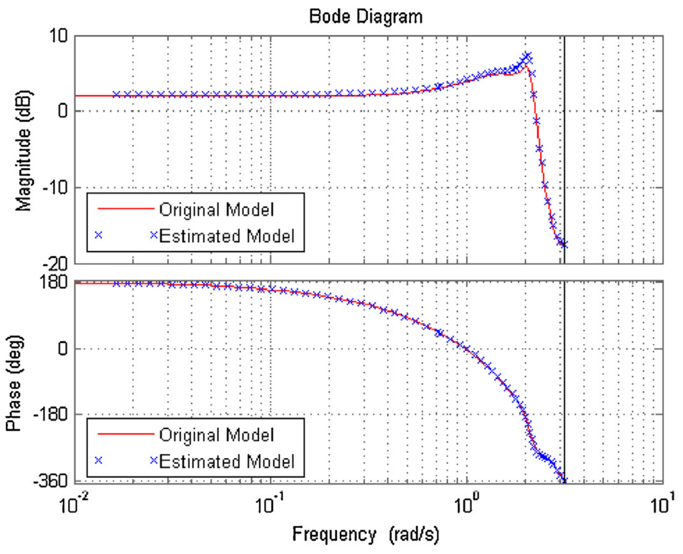

Fig. 9. Bode plot for linear model example.

Another important analysis tool that can be used in system analysis is the frequency response. Once the estimated transfer function was identified, the frequency responses for both systems were calculated and the bode plots are shown in Fig. 9. The transfer between the time-domain and frequency-domain was made possible by the proposed algorithm (i.e. it would be difficult to do similar frequency analysis without knowing the transfer function). Note that the system was appropriately identified in the frequency domain where the magnitude and phase responses matched the original frequency response of the system. It can be seen that the studied system has low pass filter characteristics with resonant peak at 100 rad/s. This information can be useful in analyzing different systems and it would have been difficult to capture using the ANN structure.

Example 2. The following model was generated to test the results for sigmoid activation function.

$$H(z) = \frac{-1.0360z^2 + 1.8780z + 0.9407}{z^3 - 0.4797z^2 + 0.1579z - 0.2143} \tag{25}$$

A two-layer ANN with four hidden nodes was used with sigmoid activation function at the hidden layer. Once the network weights

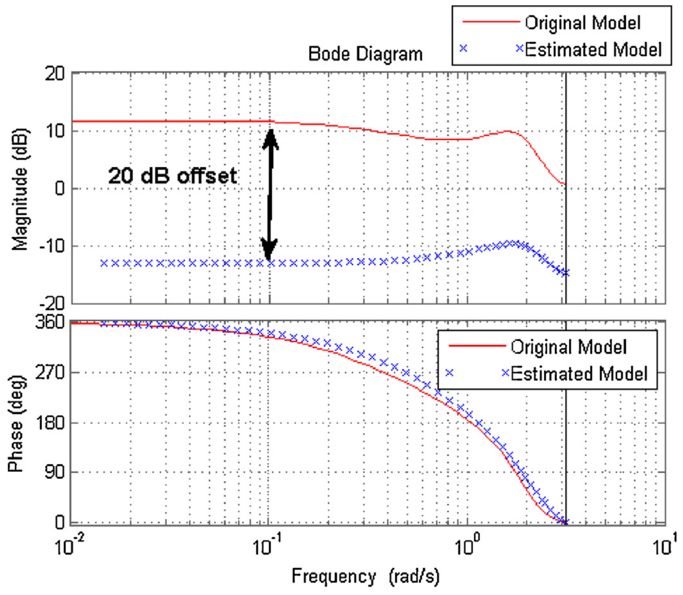

Fig. 10. Bode plot for sigmoid activation function example.

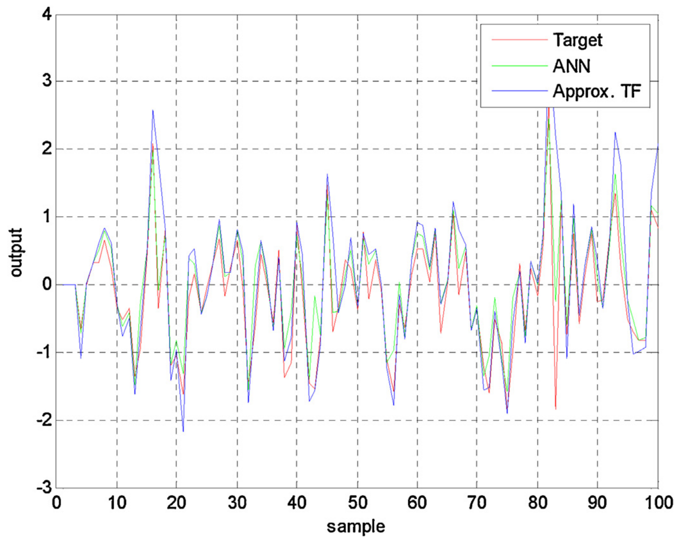

Fig. 11. Comparison results for NARMA system.

Table 7
SSE NARMA simulation results.

	Min	Max	Average	Variance
Neural network	0.0687	0.1380	0.1500	0.0030
Estimated transfer function	0.2050	1.3400	0.8270	0.7110

converged, they were converted to ARMA model and the following transfer function was derived:

$$\hat{H}(z) = \frac{-0.1235z^2 + 0.2140z + 0.0888}{z^3 - 0.1839z^2 + 0.0761z - 0.0815} \tag{26}$$

The bode plots of both systems is shown in Fig. 10 where the two systems had similar phase responses. Also, the magnitude response had similar response, but with 20 dB offset. In this case, the network's output weights converged to $wout = [0.6041, 0.1338, 1.0251, -0.3488]$. Using the offset in Table 2 for sigmoid function (i.e. $0.5\sum_{h=1}^{H}(wout_h)$) provides a 0.7071 offset which is equivalent the 20 dB offset shown in Fig. 10. This result shows that the derivations for the sigmoid functions are correct.

4.2. Nonlinear models

In the previous subsection, many simulation runs were provided to validate the proposed parameter identification algorithm's results for general linear models (i.e. ARMA models). In this section, the proposed algorithm will be further tested using two nonlinear models: Nonlinear Auto-Regressive Moving-Average (NARMA) and sinusoid with exponential term

First, a NARMA model which was used in Ref. [19] is investigated here. NARMA models use structures that are similar to ARMA models (i.e. they build models based on past inputs and outputs), but they also include multiplications between the input and output signals which produces the nonlinear terms.

Example 3. The following NARMA model is used to validate the proposed algorithm.

$$\begin{aligned} y(k) &= 0.8x(k) - 0.13x(k-2) + 0.2y(k-1) - 0.11y(k-3) \\ &\quad -0.11x^2(k-1) + 0.13y^2(k-2) - 0.18x(k-1)y(k-1) \end{aligned} \tag{27}$$

This model includes three nonlinear terms: input square, output square, and input–output multiplication. Because the ANN weights are initialized randomly, the network weights can converge to different local optima. Therefore, results for 10 simulation runs were collected and displayed in Table 7. The data size and the number of hidden nodes were fixed to 100 and 5 respectively while the order number was varied between 3, 4, and 5. Displayed results show that the proposed method causes an increase in SSE when compared to the ANN results. This is expected because the estimated transfer function is based on linear expansion of the converged ANN weights. Still, this increase is acceptable because the SSE remains small.

In order to provide a better insight into the results displayed in Table 7, the response of a 3rd order dynamic network with 5-hidden neuron were analyzed in the time-domain and compared to the original model. Random input was applied to the three models: original model (i.e. target in Eq. (27)), ANN model, and estimated transfer function (using proposed method). Note that the estimated model was able to follow the target with minimal error. Fig. 11 shows that the proposed method can provide good estimation of a linear transfer function for nonlinear models as the response of the approximated transfer function closely match the response of the original (i.e. target) transfer function.

The ANN weights were used to generate the following estimated transfer function:

$$\hat{H}(z) = \frac{0.9457Z^2 + 0.5773z - 0.0793}{Z^3 - 0.1248Z^2 + 0.0045Z + 0.3418} \tag{28}$$

Next, the impulse of the original NARMA and estimated transfer function are shown in Fig. 12. Although the estimated model has some oscillations between 5 and 10 s, but it follows the original model behavior where it starts with above 0.8 amplitude and settles down around the 15th sample.

The derivation of the transfer function provided the opportunity to do frequency-domain analysis where the bode plots are shown in Fig. 13. The bode plot can reveal important information, such as the resonance frequency, peak gain, gain margin, and phase margin. This information can help in the system analysis and in the design of appropriate controllers. These analysis and design issues are beyond the scope of this paper, but the contribution was to make such information available when needed. Such figures would have been difficult to attain from the ANN model.

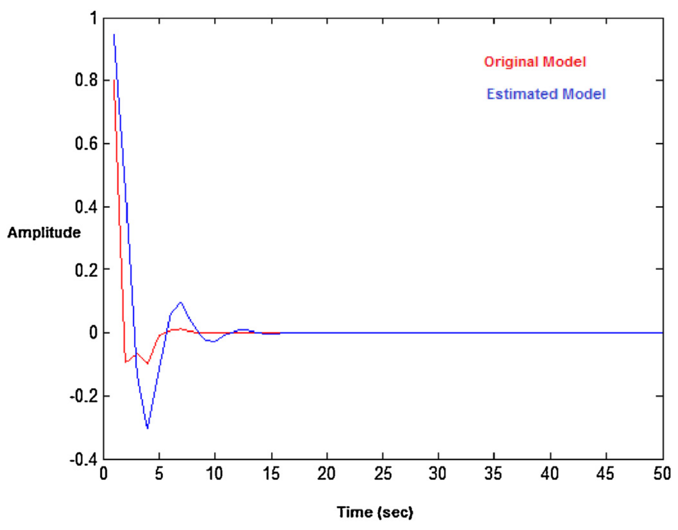

Fig. 12. Impulse responses for NARMA and estimated transfer function.

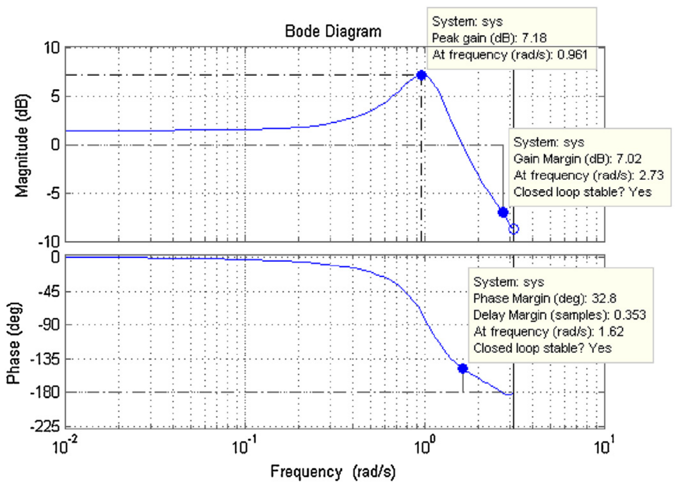

Fig. 13. Bode plots for estimated nonlinear model.

Table 8
ANN weights.

w			*v*			wout
−0.4593	−0.9077	−0.5770	0.1060	0.8303	−0.0555	−0.0743
−0.3187	−0.9928	0.8181	0.2377	−0.5629	0.6961	0.2940
0.3395	−0.7273	0.3252	−0.3043	−0.0085	−0.4472	−0.7341
0.8606	−0.8595	0.0539	0.3161	0.4057	−0.3750	1.3177
−0.8796	−0.0460	−0.8290	−0.7434	−0.1140	0.5077	0.0471

Example 4. The following nonlinear function with added noise was used.

$$y(k) = \sin(x(k)) \times e^{-0.005x(k)} + n(k) \tag{29}$$

A feedforward neural network with five hidden nodes and three unit delays was used. The ANN weights converged to the values shown in Table 8.

The formulas provided in Table 2 (with hyper tangent function and $\alpha = 2$) were used to convert the network weights into the ARMA parameters and the estimated transfer function was found to be:

$$\hat{H}_1(z) = \frac{0.7837z^3 - 0.8253z^2 + 0.0767z}{z^3 - 0.667z^2 - 0.3083Z - 0.06683} \tag{30}$$

This transfer function had one pole outside the unit circle and therefore resulted in an unstable system. The location of this pole was adjusted (using the pseudo code provided in Section 3) and moved within the unit circle and an adjusted transfer function was found as:

$$\hat{H}_2(z) = \frac{0.7837z^3 - 0.8253z^2 - 0.0767z}{z^3 - 0.564z^2 - 0.271Z - 0.06015} \tag{31}$$

Table 9
Pole locations for the Models (30) and (31).

	Poles
First model (unstable)	1.0295 $-0.1813 \pm 0.1791j$ $-0.1813 - 0.1791i$
Second model (stable)	0.9265 $-0.1813 \pm 0.1791j$ $-0.1813 - 0.1791i$

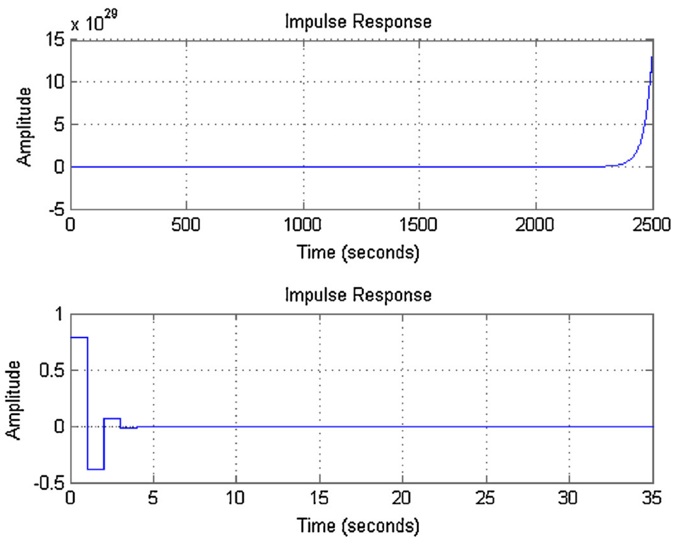

Fig. 14. Impulse response of the estimated transfer function.
(a) Unstable model (b) Stable model.

The pole locations of both transfer functions are displayed in Table 9. Note that only the first pole was adjusted from 1.0295 to 0.9265.

Once the transfer function was identified, the system was further analyzed by calculating the impulse response as shown in Fig. 14. Note that the first estimated model produced an unstable response as the system's response increased to infinity while the adjusted model produced an impulse response that settled around the 4th sample.

The network and the estimated transfer function were compared with the original nonlinear function as shown in Fig. 15. The estimated transfer function followed the target behavior well.

This example showed that the proposed algorithm was able to use the ANN weights to calculate a stable model (with minimum error) that approximated a nonlinear model with added noise.

5. Conclusion

Researchers have used neural network models for system identification applications with good success. However, these network models hide the parametric information of the system within their structures. In this paper, a clear mathematical relationship between the network weights and the system parameters was established in order to calculate estimated transfer functions. Parametric identification can provide valuable information and insight about the system's behavior such as the pole/zero locations and bode plots.

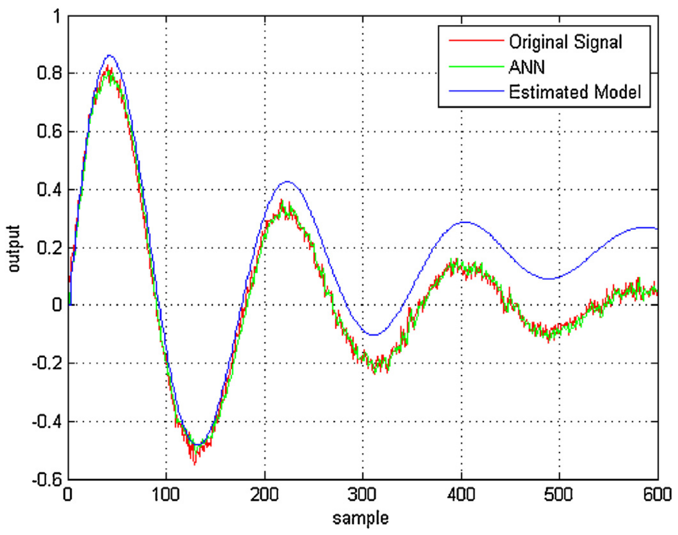

Fig. 15. Nonlinear system output response.

In order to make the presented work valuable for other researchers, an easy-to-follow algorithm, NN2TF, was derived, analyzed, and validated. This algorithm has the capability of transforming the weights of a multi-layer neural network into an accurate transfer function and can be used by researchers to capture important information about their identified systems.

Simulation runs for four different models were used to validate the algorithm's results. First, two linear models were used to validate the accuracy of the estimated transfer functions (using NN2TF) as their time and frequency responses were compared to the original and ANN models. Then, a nonlinear model was generated to test the capability of the proposed algorithm to generate accurate transfer functions. All results showed that the estimated transfer function was able to mimic the behavior of the original models with minimum error. Finally, a third nonlinear model was used to show that the algorithm was able to adjust its converged parameters in order to produce stable models.

Acknowledgements

I would like to thank Prof. Victor DeBrunner for introducing me to system identification during my MS studies, Prof. Theodore Trafalis for introducing me to neural networks during my PhD studies, and Prof. Munther Baker for proof-reading this manuscript.

References

[1] S. Haykin, Neural Networks and Learning Machines, 3rd edition, Prentice Hall, 2008.
[2] G.P. Liu, Nonlinear Identification and Control: A Neural Network Approach, Springer, 2001.
[3] M. Norgaard, O. Ravn, N.K. Poulsen, L.K. Hansen, Neural Networks for Modelling and Control of Dynamic Systems: A Practitioner's Handbook, Springer, 2003.
[4] A. Zilouchian, M. Jamshidi, Intelligent Control Systems Using Soft Computing Methedologies, CRC Press, 2001.
[5] H. Demuth, M. Beale, M. Hagan, Neural Network Toolbox 6 User's Guide Mathworks, 2009.
[6] R. Isermann, M. Munchhof, Identification of Dynamic Systems: An Introduction with Applications, Springer, 2011.
[7] M.O. Efe, K. Kaynak, A comparative study of neural network structures in identification of nonlinear systems, Mechatronics 9 (1999) 287–300.
[8] G.P. Liu, V. Kadirkamanathan, S.A. Billings, On-line identification of nonlinear systems using Volterra polynomial basis function neural networks, Neural Netw. 11 (December (9)) (1998) 1645–1657.
[9] I. Gabrijel, A. Dobnikar, On-line identification and reconstruction of finite automata with generalized recurrent neural networks, Neural Netw. 16 (2003) 101–120.
[10] H.K. Sahoo, P.K. Dash, N.P. Rath, NARX model based nonlinear dynamic system identification using low complexity neural networks and robust $H\infty$ filter, Appl. Soft Comput. 13 (2013) 3324–3334.
[11] R. Coban, A context layered locally recurrent neural network for dynamic system identification, Eng. Appl. Artif. Intell. 26 (2013) 241–250.
[12] J. Deng, Dynamic neural networks with hybrid structures for nonlinear system identification, Eng. Appl. Artif. Intell. 26 (2013) 281–292.
[13] I. Darus, A. Al-Khafaji, Non-parametric modelling of a rectangular flexible plate structure, Eng. Appl. Artif. Intell. 25 (2012) 94–106.
[14] H.G. Han, L.D. Wang, J.F. Qiao, Efficient self-organizing multilayer neural network for nonlinear system modeling, Neural Netw. 43 (2013) 22–32.
[15] S.L. Xie, Y.H. Zhang, C.H. Chen, X.N. Zhang, Identification of nonlinear hysteretic systems by artificial neural network, Mech. Syst. Signal Process. 34 (2013) 76–87.
[16] M. Khashei, M. Bijari, An artificial neural network (p, d, q) model for time series forecasting, Expert Syst. Appl. 37 (2010) 479–489.
[17] G.P. Zhang, Time series forecasting using hybrid ARIMA and neural network model, Neurocomputing 50 (January) (2003) 159–175.
[18] C.F. Fung, F.A. Billings, H. Zhang, Generalised transfer functions of neural networks, Mech. Syst. Signal Process. 1 (6) (1997) 843–863.
[19] K. Chon, R. Cohen, Linear and nonlinear ARMA model parameter estimation using an artificial neural network, IEEE Trans. Biomed. Eng. 44 (March (3)) (1997).
[20] J. Lopez, E. Caicedo, Parametric identification using multilayer perceptron, International Conference on Industrial Electronics and Control Applications (2005).
[21] Y. Chen, Z. Chen, A neural-network-based experimental technique for determining z-transfer function coefficients of a building envelope, Build. Environ. 35 (2000) 181–189.
[22] M. Fei, J. Zhang, H. Hu, T. Yang, A novel linear recurrent neural network for multivariable system identification, Trans. Inst. Meas. Control 28 (3) (2006) 229–242.
[23] T.A. Tutunji, Approximating transfer functions using neural network weights, in: Proceedings for the 4th International IEEE EMBS Conference on Neural Engineering, Antalya, Turkey, 2009.
[24] T.S. Soderstrom, P.G. Stoica, System Identification, Prentice Hall, 1989.
[25] T.A. Tutunji, M. Molhem, E. Turki Mechatronic, Systems identification using an impulse response recursive algorithm, Simul. Model. Pract. Theory 15 (2007) 970–988.
[26] G. Franklin, J. Powel, M. Workman, Digital Control of Dynamic Systems, 3rd edition, Ellis-Kagle Press, 1998.
[27] M.T. Hagan, M.B. Menhaj, Training feedforward networks with the Marquardt algorithm, IEEE Trans. Neural Networks 5 (6) (1994).
[28] Z.J. Fu, W.F. Xie, W.D. Luo, Robust on-line nonlinear systems identification using multilayer dynamic neural networks with two-time scales, Neurocomputing 113 (2013) 16–26.
[29] P. Gil, J. Henriques, A. Cardoso, A. Dourado, On affine state-space neural networks for system identification: global stability conditions and complexity management, Control Eng. Pract. 21 (2013) 518–529.

Tarek A. Tutunji is currently serving as Dean of Development and Quality at Philadelphia University (PU) in Jordan. He is a founding member of the Mechatronics Engineering Department at PU, served as the Department Chair for six years, and as the Dean of Scientific Research and Graduate Studies for two year. He has experience in manufacturing and design development where he worked for four years as manufacturing engineer with Halliburton in Texas and two years as design developer with Seagate in Oklahoma. He has a Ph.D. in industrial engineering and MS in electrical engineering, both from University of Oklahoma, USA. His research interests is in the control and identification of mechatronic systems.